On Programming Information Agent Systems – An Integrated Hotel Reservation Service as Case Study*

Yun Ding, Heiner Litz, Rainer Malaka, and Dennis Pfisterer

European Media Laboratory,
Schloss-Wolfsbrunnenweg 33, 69118 Heidelberg, Germany
{Yun.Ding,Heiner.Litz,Rainer.Malaka,Dennis.Pfisterer}@eml.villa-bosch.de

Abstract. This paper presents our integrated hotel reservation service. Using it as a case study, we discuss the design and implementation of agent-based information systems. Taking a system as a whole, we consider not only information agents but also their interface to human users and external information sources. In particular, our focus is on the interaction behavior, which can be observed both in interactions between agents and in interactions between agents and these interface components. We show that both kinds of interaction are coordinated by the same protocol. Using our implemented hotel reservation service system, we illustrate exemplarily how this understanding can be used to systematically design and validate interaction mechanism. We explore the possibility to facilitate the rapid prototyping of information agent systems using an interaction behavior editor. Moreover, by giving insight into some details of our hotel service system, we exemplify where the difficulties in implementing information agent systems are and thus infrastructural support are desirable.

1 Introduction

The rapid growth and popularity of the World Wide Web have largely eased the access to information. At the same time, it is increasingly difficult for human people to collect, filter, evaluate and really use the vast amount of information. *Information agents* are software programs that gather information from various, heterogeneous and distributed sources on behalf of their human users to achieve their goals. For instance, they filter away irrelevant information, integrate information from different sources, proactively monitor and search for more related information, and eventually present them in an appropriate way to their users [9]. In the past, information agents have been developed for different domains such as shopping (Menczer et. al. present a good survey over existing shopping

* The work is funded by the Klaus Tschira Foundation, the German Federal Department for Education, Science, Research, and Technology (BMBF) in the scope of project EMBASSI under FKZ 01 IL 904 D2, and the Agentcities.NET under ACNET.02.32.

M. Schillo et al. (Eds.): MATES 2003, LNAI 2831, pp. 50–61, 2003.
© Springer-Verlag Berlin Heidelberg 2003

agents in [20]), hotel/restaurant reservation [26,22], or portfolio management in the financial domain [9].

Information agents are just one part of an agent-based information system that additional consists of interface agents and middle agents. Interface agents interact with human users to receive their requests and deliver results. On behalf of human users, interface agents interact with information agents. Middle agents facilitate the location of agents in an open system.

In the scope of the projects EMBASSI [12] and Agentcities [1], we have developed an information system for the hotel service domain, which is called an integrated hotel reservation service. Currently, a number of hotel searching and reservation services are available on the web offering proprietary on-line searching and booking capabilities. Users have to browse through different web sites and formulate their requests anew for each of the hotel reservation systems, which normally have different user interfaces and use different terms for identical parameters. Our hotel service system uses several different vendor web sites to parallelly process a search request while offering a unified user interface and terminology.

In contrast to other related works which focus on the modeling, designing and implementation of information agents [10,8], or their low-level skills such as information extraction [18], filtering and integration, we consider the *design and implementation of interaction behavior* in an information system. Taking a system as a whole, we take into account not only information agents but also their interface to human users and external information sources. We made the following observation: When a certain interaction protocol (for example, a contract-net protocol specified by the FIPA specifications [14] for inter-agent communication) is chosen to achieve a user's goal, it coordinates not only the interaction between an interface agent and information agents but also guides the interaction between the building components of interface and information agents. This understanding is useful for systematically designing agent components. Going a step further, it might be used to construct toolkits which allow rapid prototyping of information systems and even (semi-)automatical programming. We briefly present the architecture and implementation status of our integrated hotel reservation service in section 2. Using it as a case study, the subsequent sections consider the aspects of designing interaction behavior, using ontology and agent collaboration in information systems respectively. We both describe implemented concepts for our hotel reservation service and discuss these aspects when abstracting from the particular hotel service domain. In section 6 we compare our work with related work. Section 7 sums up and points out future research directions.

2 The Case Study: Integrated Hotel Reservation Service

The architecture of our integrated hotel reservation service is illustrated in figure 1. It consists of three main parts, user interface agents, hotel service agents as information agents, and a hotel service manager as an information broker.

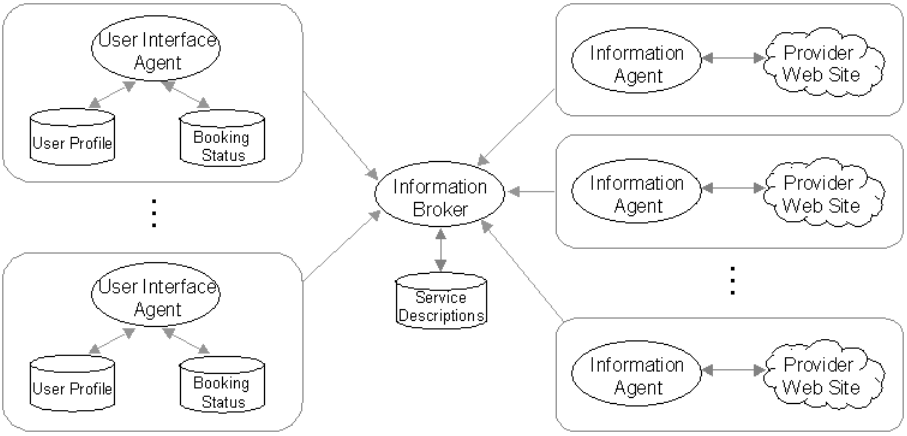


Fig. 1. System architecture of our integrated hotel reservation service

- *User interface agents:* Each user interface agent represents a human user. It trustfully keeps information such as user profiles and reservations made. On behalf of its user, it contacts hotel service agents for searching and booking a hotel or cancellation of an earlier reservation. A graphical interface (GUI) is used to navigate the interactions between the user interface agent and its user.
- *Hotel service agents:* Each hotel service agent encapsulates a web site, which provides hotel reservation services. In an open environment as the World Wide Web is located in, new web sites and thus their associated hotel service agents can dynamically join into or leave the system.
- *Hotel service manager:* The hotel service manager is designed to enable user interface agents to find the matching hotel service agents, which can handle their requests. It acts as a matchmaking middle agent [29] between them. Hotel service agents advertise and de-advertise their capabilities to the hotel service manager, while user interface agents query for matching hotel service agents to serve the requests of human users.

We implemented a prototype of integrated hotel reservation service on our agent platform *RAJA* [11]. RAJA is a FIPA [14] compliant agent platform, which is implemented on top of the well-known FIPA implementations *FIPA-OS* [15] and its lightweight version for small devices *Micro FIPA-OS* [21] as well as *JADE* [3]. The current prototype has hotel service agents for the vendors HRS (www.hrs.de), HRN (www.hoteldiscount.fr), Travelocity (www.travelocity.com) and Budgethotel (www.budgethotel.com).

3 Designing Interaction Behavior

Figure 2 shows a simplified system architecture of an agent-based information system. The interface agent interacts with a human user through a graphical

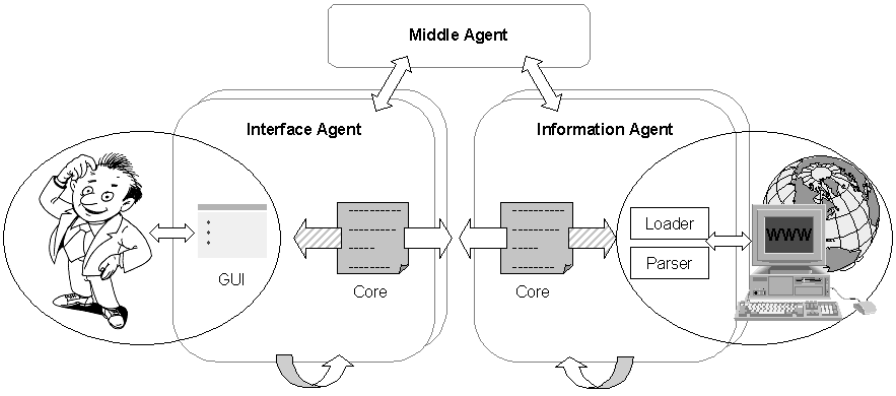


Fig. 2. Architecture of an information system and overview of interactions

interface (GUI). Information agents retrieve and parse information from the World Wide Web through their site-specific components which are called loader and parser respectively. A loader has two types of methods. One of them are used to map user-defined criteria to fields of vendor-provided forms and can be performed off-line, while the others need on-line connections to the vendor site. Methods of the second type are invoked to submit a form filled in (e.g., a form to search for hotels) or a decision from the user (e.g., accept or reject a booking proposal). Upon receiving results from vendors, which are mostly HTML-based, the task of a parser is to extract relevant information such as the names, addresses and prices of hotels from them. The core component of an interface or information agent implements the logical part/functionality of an agent. It is reusable in the sense that it is independent from the represented users and the associated external information sources. The blank arrows symbolize inter-agent interactions and interactions between an agent and its environment (i.e., a human being or an external source), while the two gray, stripy arrows symbolize interactions between the components inside an agent.

At the level of inter-agent interaction, interface agents represent human users, while information agents encapsulate external resources. However, taking the viewpoint of a user and a vendor-specific site into account, interface and information agents become new roles: (1) From the user's or the GUI's viewpoint, the associated interface agent or its core component represents the "information world", which may include several information agents and other agents. (2) From the viewpoint of a vendor-specific site or its associated loader and parser, an information agent or its core component retrieves and extracts information from it on behalf of a user. These two points consider intra-agent interactions, i.e., interactions between the GUI and the core component of an interface agent or interactions between the loader/parser and the core component of an information agent. It is not difficult to see, that the protocol for the inter-agent interaction should also guide the intra-agent interaction, which is achieved through method invocations between the components. According to the steps specified in an in-

teraction protocol, the methods for the interactions between the components can be systematically designed. In the following, we exemplify this interaction behavior in our integrated hotel reservation service.

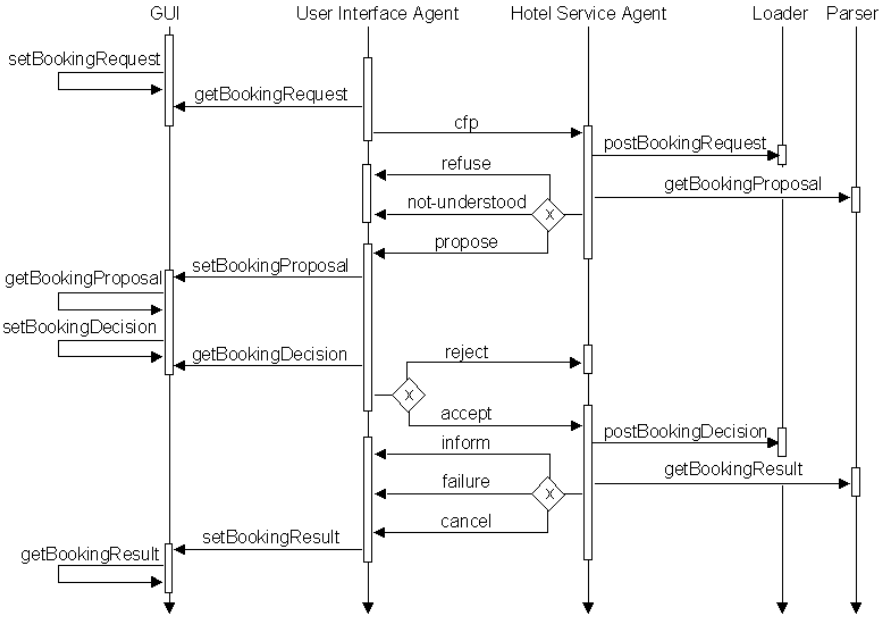


Fig. 3. Interaction diagram

Representing a vendor-specific web site, a hotel service agent interacts with a user interface agent to provide the services offered by the vendor. The interactions follow some protocols. In our system, the action "*search-hotel*" follows the FIPA-request protocol, while the actions "*book-hotel*" and "*cancel-reservation*" follow the FIPA-contract-net protocol [14]. Each action will be performed by a specialized task in its own thread. Therefore, multiple requests can be parallelly handled. For a *book-hotel* action, figure 3 shows the interaction diagram (using the notation in [23]) for the interactions among a GUI, its owning user interface agent, a hotel service agent, and its loader and parser component. The communication between a user interface agent and hotel service agents are asynchronous, since participants must not be blocked waiting for incoming messages. This enables a hotel service agent to parallelly handle requests from different users. In contrast, the communication between the components of an agent is synchronous, i.e., the `get` methods are blocking methods.

It is easy to see that the methods of the components GUI, loader and a parser are systematically derived from the interaction protocol obeyed and the domain specific ontology. This interaction behavior is useful for the rapid and systematical design and validation of interaction mechanisms for agent-based information systems. Going a step further, it might be used to construct toolkits which allow rapid prototyping of information systems and even semi-automatical

programming. We are developing an *Interaction Behavior Editor*. It should allow programmers to define an interaction protocol by specifying its steps. Each step is characterized by

- a *name* and a *type*, which indicate the domain-specific issue concerned (e.g., booking_requests, proposals, decisions, and booking_results) and its data type respectively,
- a *role*, which specifies whether it is taken by the initiator or the responder of the interaction, and
- a *protocol*, which specifies the allowed actions for this step (e.g., propose, refuse, and not-understood for the second step of a contract-net protocol). The actions are related to each other by "exclusive-or", and
- its *next step*, which points to the next step of the interaction protocol.

Currently, we specify the interaction protocol in XML (an excerpt is shown in figure 4). Then, we use a set of XSL (eXtensible Stylesheet Language) style sheets to transform it into Java classes for

- the *GUI component* of the interface agent with `set` and `get` methods, whereas the `get` methods are blocking methods, and
- the *loader and parser component* for the information agent. A loader acts as an initiator of an interaction and provides `post` methods, while a parser acts as a responder and offers `get` methods. Application programmers should overwrite the abstract `post` and `get` methods for each vendor site.
- *agent behaviors/tasks* such as contract-net-initiator or contract-net-responder behavior provided by the JADE agent framework [3]. For example, an agent behavior as a contract-net-initiator offers methods `handlePropose(incoming_msg)`, `handleRefuse` and `handleNotUnderstood`, which should be overwritten by programmers.

To allow information agents to communicate with other information sources such as databases, specialized loaders can be constructed.

The effort we put into designing interaction behavior for components, which interact with the "outside world" (i.e., human user or external sources), is justified when intense feedback from the users is required to perform an action such as our *book-hotel* or *cancel-reservation* action. For simple one-shot queries, the interaction pattern is straightforward and can be easily hand-coded.

4 Using Ontology

Ontologies provide a shared and common understanding of a domain to enable the communication between people and heterogeneous and distributed application systems [17]. For a particular domain, an ontology defines its concepts (e.g., a concept of hotel for the domain of hotel service) and their associated attributes (e.g., each hotel has a name, an address, prices and a comfort category), as well as the relationships between the concepts. Using ontology is twofold. Firstly, ontologies facilitate the building of knowledge-based systems. Upon defining concepts and their relationship of a domain, individual instances of concepts can

```
<InteractionBehavior name="BookingHotel">
    <Steps>
      <step name="booking_request" role="initiator" type="HotelLanguage"
                                                  next="booking_proposal">
         <protocol>  <action name="CFP" end="false"/> </protocol>
      </step>
      <step name="booking_proposal" role="responder" type="HotelLanguage"
                                                   next="booking_decision">
         <protocol>
            <choice>
              <action name="refuse" end="true"/>
              <action name="not_understood" end="true"/>
              <action name="propose" end="false"/>
            </choice>
         </protocol>
       </step>
        ......
     </Steps>
</InteractionBehavior>
```

Fig. 4. Specification of interaction behavior in XML: an excerpt

be created and stored in a knowledge base. The resulted knowledge base can be used with problem-solving methods to answer questions (e.g., find all hotels in the region Heidelberg with prices lower than 100$ per night) and solve problems. Secondly, for agent-based information systems, ontologies are needed both for extracting information from different sources, which often use different words for the same piece of information and for the inter-agent communication. Agents located inside an agent platform efficiently communicate via Java objects as content of messages. For inter-platform communication, string-based content languages such as KIF, SL or XML are widely used. Figure 5 outlines the steps and supporting software tools to use ontology as described above.

The ontology editor allows users to create both ontologies and individual instances of concepts into a knowledge base. The knowledge base is used by a reasoning system such as the JTP [16] or FaCT [13], while the ontologies will be translated into XML Schema to be used for the exchange of data [17] (e.g., between agents). Supports for the data exchange include (1) the automatical creating of Java classes from a XML Schema, and (2) the automatical conversion of Java objects into strings of a particular content language and vice versa.

A plug-in of the ontology editor *Protégé* called *beangenerator* [2] creates schemas and Java classes from ontologies defined using Protégé [27]. However, they can only be used within the JADE agent framework [3]. The two available XML Schema plug-ins of Protégé [28] do not produce XML Schemas, which can be subsequently used to automatically create Java classes. Apart from them, *oil2xsd* [24] converts ontologies written in OIL into XML Schema, and the source generator of the Castor project [5] automatically creates Java classes from a XML Schema. Putting it together, ontologies written in OIL can be transformed into

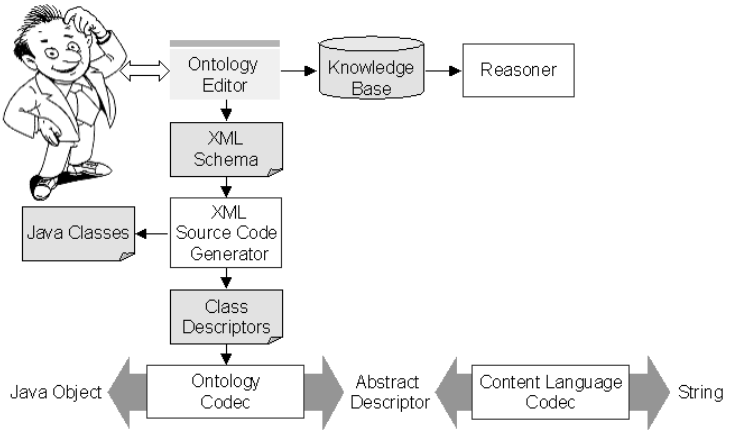


Fig. 5. Steps and software tools for the using of ontologies

Java classes using *oil2xsd* and the source generator of Castor. However, we decided to develop our ontology (or be more correct, the structure of data to be exchanged between agents) direct in XML Schema without an ontology editor, since we retrieve information from external sources instead of keeping an own knowledge base.

For the automatical conversion between Java objects and strings, we use a solution, which is slightly different from the one applied by JADE [4]. *Abstract descriptors* serve as intermediate objects between Java objects and the corresponding language-specific strings. Each Java object can be represented by an abstract descriptor that includes a number of named slots holding the values of its attributes. A *content language codec* is responsible for the conversion between strings and abstract descriptors, while an *ontology codec* converts a Java object into an abstract descriptor and vice versa. The ontology codec binds each Java class to a schema, which defines the structure of the Java class. In our approach, we use the schema encoded in the *class descriptors*, which are produced by the Castor source code generator together with the Java classes. A class descriptor provides methods to set and get the value of an attribute of the object. Therefore, we do not have to use the time-consuming Java Reflection API to inspect the Java objects, as it is done in JADE.

5 Agent Collaboration

Combinatively, five kinds of agent collaboration are conceivable in an information system. They are 1) collaboration between user interface agents, 2) between information agents, 3) between middle agents, 4) between interface and information agents, and 5) between interface, middle and information agents. The collaboration between user interface agents facilitates opinion-based filtering as used by the integrated restaurant service *IRES* [22]. When a user interface agent discovers a new restaurant or is not sure about a recommendation, it asks other

user interface agents for their opinion. Trust values are used to divide agents in reliable friends and unreliable ones. Using collaboration between information agents, an information agent can delegate a request or part of a request to other information agents, if it is unable to serve the request alone [26]. Using several cooperating middle agents ensures load balancing and avoids single point of failure. Our integrated hotel reservation service employs the last two kinds of collaboration.

As a matchmaking middle agent, a hotel service manager matches *requests* from user interface agents to *capabilities* of registered hotel service agents. A challenging task is to define models of requests and capabilities. According to [10], a description of capabilities/service should consist of both basic service characteristics which are domain independent and service-specific characteristics. Our *hotel service description* which characterizes the service provided by a vendor site consists of the following criteria. The first three are basic characteristics, while the remaining ones are service-specific.

- *Popularity of a site:* summarizes how frequent human users have made use of its service, for instance, made a reservation through it.
- *Reliability:* is the likelihood of service failures,
- *Cost:* indicates the price-performance ratio it provides.
- *Service coverage and bias:* for example, the service of a vendor site may cover hotels located in Europe, Asia or worldwide. A vendor may claim a worldwide coverage but is specialized for hotels in Europe. The number of hotels found in different locations gives a hint on the coverage and bias.
- *Description of supported actions:* hotel services are provided in terms of several supported actions such as *search-hotel*, *book-hotel*, *cancel-reservation* and *change-reservation*. Each type of action is characterized by its vendor-defined cost (e.g., changing a reservation will be charged for 10$.) and its average duration.

Some characteristics in a hotel service description can be easily acquired from the vendor web site. They are, for example, the supported actions and their vendor-defined cost. Other characteristics like the duration of an action, reliability, popularity and cost of a site must be measured at runtime by user interface agents or evaluated by their associated human users. A hotel service manager stores hotel service descriptions provided by hotel service agents and updates them using the feedback from user interface agents. Each hotel service agent registers itself to the hotel service manager using a *hotel service agent description*, which includes the name of the agent, the address of the vendor site it encapsulates and the service description of the site. We model the requests from user interface agents using a *hotel service request description*. It includes

- the name of the desired action (e.g., *book-hotel*),
- the location of the searched hotels which is used to match against the service coverage and bias of hotel service agents, and
- search constraints. They specify the maximal number of expected results and preferences of the users. In response to the prevailing resource situation, e.g.,

in case of lower bandwidth, a resource-aware user interface agent would like to contact only one or two but not five or six hotel service agents. Preferences is a list of criteria on hotel service (e.g., {*reliability, price, popularity*}) which are ordered by descending importance for the user. They are used to rank the results.

6 Related Work

The bed and breakfast reservation service [26] targets the same domain as our integrated hotel service. Their approach differentiates from ours in two main aspects. Firstly, they do not use the World Wide Web as information source. Instead, hotel information is delivered by hotel providers and kept in the databases of the so-called B+B agents (Bed and Breakfast agents). Secondly, the B+B agents collaborate in order to satisfy users' requests. Whenever a B+B agent is not able to serve the user, it delegates the request to other B+B agents. In our system, a user interface agent broadcasts the user's request to several hotel service agents and gathers the results. We give human users more control over which hotel service agents to contact. For the same reason, users are asked to evaluate the services provided by different vendors.

Decker et. al. presented in [8] an architecture for information agents and described a set of reusable behaviors/tasks of these agents such as advertising, information monitoring and cloning. Rapid construction of new classes of information agents can be enabled through the reuse and combination of these existing behaviors. Their work on agent behaviors is complementary to our work on interaction behaviors.

Recently, the OpenTravel Alliance (OTA) publishes their specifications for the interoperability of travel services (e.g., hotel, flight and holiday booking or car rental) [25]. The specifications define message contents that are transferred among travel services in XML Schema. As part of future work we intend to compare our schema with the OTA schema.

In the advent of web services [19], languages and ontologies such as WSDL [6] and DAML-S [7] have been defined for describing the properties and capabilites of web services. Our *service description* and *hotel service agent description* are used by the hotel service manager to *locate* the hotel service agents, but not to invoke or activate their service. They are comparable to the service profile part of DAML-S. It is conceivable that both a single hotel service agent and the integrated hotel reservation service can participate in an open environment, which web services focus on.

7 Conclusions and Future Work

In this paper, we presented our information agent system, an integrated hotel reservation service system. Using it as a case study, we discussed the design and implementation of information agent systems.

When considering "periphery" of information agent systems such as interface to human users and external information sources, most of previous works focus on the design of human machine interface or on the task of information retrieval. They take the interaction of these components to the rest of the system as granted. However, our experience in implementing our hotel reservation services showed that it is worth investigating the *interaction behavior* when taking an agent-based information system *as a whole*. Although interactions between agents and between components of an agent are achieved in different ways, they are guided by (almost) the same protocol. Using our hotel reservation system as an example, we illustrated how this apparently trivial observation helps systematically designing and validating interaction mechanism. Then, we illustrated how to enable rapid prototyping of information systems using source code generation with XSL.

Apart from the aspect of interaction, we described how ontologies can be used in information-based agent systems. Our experience showed that it is very time-consuming to find the appropriate software tools to use ontologies. We proposed the steps to be go through to use ontologies and evaluated software tools, which automate these steps. To enable collaboration between interface and information agents through matchmaking, we have developed models, which characterize the capabilities of hotel service agents and requests issued by the users. Feedback from the users (or their associated agents) are used to dynamically adapt the capabilities modeled.

Programming agent-based information systems is inherently interdisciplinary. We take our hotel reservation service system, which is still work in progress as playground to investigate the broad range of aspects arising in information systems. For future work, we plan to develop an infrastructure, which supports the design and software reuse of information agent systems.

References

1. Project agentcities. http://www.agentcities.org.
2. *Bean generator*. http://gaper.swi.psy.uva.nl/beangenerator/content/main.php.
3. F. Bellifemine, A. Poggi, and G. Rimassa. JADE - A FIPA-compliant agent framework. In *Proc. The Practical Application of Intelligent Agents and Multi-Agents (PAAM 99)*, pages 97–108, 1999. http://sharon.cselt.it/projects/jade/.
4. G. Caire. *JADE Tutorial: Application-Defined Content Languages and Ontologies*, 2002.
5. *Castor - The Source Generator*. http://castor.exolab.org.
6. E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. *Web Services Description Language (WSDL) 1.1*, 2001. http://www.w3.org/TR/2001/NOTE-wsdl-20010315.
7. DAML-S Coalition. DAML-S: Web Service Description for the Semantic Web. In *Proc. of the first International Semantic Web Conference (ISWC)*, 2002.
8. K. Decker, A. Pannu, K. Sycara, and M. Williamson. Designing Behaviors for Information Agents. In *Proc. of the first Conference on Autonomous Agents (Agents'97)*, 1997.

9. K. Decker and K. Sycara. Intelligent Adaptive Information Agents. *Journal of Intelligent Information Agents*, 9:239–260, 1997.
10. K. Decker, K. Sycara, and M. Williamson. Modeling Information Agents: Advertisements, Organizational Roles, and Dynamic Behavior. In *Proc. of the AAAI-96 Workshop on Agent Modeling*, 1996.
11. Y. Ding, C. Kray, R. Malaka, and M. Schillo. RAJA-A Resource-Adaptive Java Agent Infrastructure. In *Proc. of the 5th International Conference on Autonomous Agents (Agents 2001)*, 2001.
12. Project EMBASSI. http://www.embassi.de.
13. The FaCT System. http://www.cs.man.ac.uk/ horrocks/FaCT/.
14. *Foundation for Intelligent Physical Agents. Specifications*, 2000. http://www.fipa.org.
15. FIPA-OS, 1999. http://fipa-os.sourceforge.net.
16. JTP: An Object-Oriented Modular Reasoning System. http://ksl.stanford.edu/software/jtp/.
17. M. Klein, D. Fensel, F. van Harmelen, and I. Horrocks. The relation between ontologies and xml schemas. *Electronic Trans. on Artificial Intelligence*, 2001. Special Issue on the 1st International Workshop "Semantic Web: Models, Architectures and Management".
18. N. Kushmerick and B. Thomas. Adaptive information extraction: Core technologies for information agents. In *Intelligent Information Agents R&D in Europe: An AgentLink perspective*, 2002.
19. S. McIlraith, T. Son, and H. Zeng. Semantic Web Services. *IEEE Intelligent Systems*, 16(2):46–53, 2001.
20. F. Menczer, W. Street, N. Vishwakarma, A. E. Monge, and M. Jakobsson. IntelligShopper: A Proactive, Personal, Private Shopping Assistant. In *Proc. of the first International Conference on Autonomous Agents & Multiagent Systems (AAMAS'02)*, 2002.
21. Micro FIPA-OS, 2001. http://www.cs.Helsinki.FI/group/crumpet/mfos/.
22. M. Montaner, B. Lopez, E. Acebo, S. Aciar, and I. Cuevas. *IRES: On the Integration of Restaurant Services*, 2002. Available as http://arlab.udg.es/GenialChef.pdf.
23. J. Odell, H. V. D. Parunak, and B. Bauer. Extending UML for Agents. In *Proc. of the Agent-Oriented Information Systems Workshop (AOIS) at the 17th National conference on Artificial Intelligence (AAAI)*, 2000.
24. *OIL to XML Schema Transformator.* http://savannah.nongnu.org/projects/oil2xsd/.
25. *OpenTravel Alliance Specification Version 2002B*, 2002. http://www.opentravel.org/2002b.cfm.
26. J. Padget and W. Barbera-Medina. A bed and breakfast reservation service. In *Proc. Workshop on Agentcities: Challenges in Open Agent Environments, held in conjuction with the first International Conference on Autonomous Agents & Multiagent Systems (AAMAS'02)*, 2002.
27. *The Protégé Project.* http://protege.stanford.edu/.
28. *Protégé Plug-ins Library.* http://protege.stanford.edu/plugins.html.
29. H. Wong and K. Sycara. A Taxonomy of Middle-agents for the Internet. In *Proc. of the fourth International Conference on Multi-agent Systems*, pages 465–466, 2000.